A Database Forensic Approach to Detect Tamper Using B+-Trees

Tanushree Shelare
Student, Information Technology
Maharashtra Institute of Technology
Pune, India
tanushree.shelare@gmail.com

Varsha Powar
Assistant Professor, Information Technology
Maharashtra Institute of Technology
Pune, India
varsha.powar@mitpune.edu.in

Abstract **–** Data is the most valuable thing today from an individual to an organization. Because of its large volume and useful information database contains, it is vulnerable and are more prone to attacks. Database Forensic is a new and important field which helps to analyze, identify attacks made in the database and make provisions to prevent databases from such attacks by making use of Database Forensic tools. Few tools are available for database forensics which is time consuming. In this paper, I will show how B+-trees and database forensic is related. We will define the efficient use of B+-trees in database forensic. B+-trees are data structures which are widely used in storage engines can explore history. B+-trees are fast indexing method which reduces time of a database forensic tool and increase performance.
Keywords*: forensic, B+-trees, database.*

*****

I. INTRODUCTION

Digital forensics is a branch of forensic science encompassing the recovery and investigation of material found in digital devices, often in relation to computer crime [12].

The branches of digital forensics are:

- Computer forensics: Computer forensics deals with computers. It finds evidence from storage medium or electronic document.
- Mobile device forensics*:* Mobile device forensicsdeals with mobiles. It reecovers and extract the data which can serve as evidence from a mobile device. Investigations can be done by analyzing call data and communications through SMS or email. GPS based mobile devices helps in finding location information.
- Network forensics: Network forensic deals with the computer network.Computer network traffic both local and WAN/internet, are monitored and analysed in search of evidence.
- Forensic data analysis: Structured data are examined and analysed to find patterns of suspicious activities under forensic data analysis.
- Database forensics: Database systems have become the integral part of today's society. The information databases systems contain are sensitive to organization. The organization must be aware of any tampering done with the database in the organization. To protect organizational database, efforts have been taken. But still records are breached. Database forensic is the branch of digital forensics which relates to the information found on the database. Database forensics aim is to find what happen and when and to prevent unauthorized access. Database attacks can be detected and analyzed by database forensic. Database Forensic is an emerging and new field in research area. Very few tools and literature is available till date. Traditional digital investigations always excluded databases even if evidence can be obtained from them. The field is still in its early years but it is an important part of many investigations due to the increased volume of information. This information may be helpful in solving different crimes and the large number of risks associated with the information stored on many databases. The ability to retrace the operations performed on a database and reconstruct deleted or compromised information on the database is of great importance. There should be some mechanism by which database transactions could be monitored continuously. So a track of unauthorized accesses can be kept. By keeping the track of the users activities the culprit can be held responsible.

Figure 1 shows the database forensic model which includes three major steps;

- First step involves acquisition of raw data. Data can be obtained from multisource. It can be clinical, operational or financial data.
- Second step involves proper integration and organization of data .
- Third and important step is to work out with data, i.e. to analyze the data and take appropriate action. This can be done by applying proper data mining or inference rules.

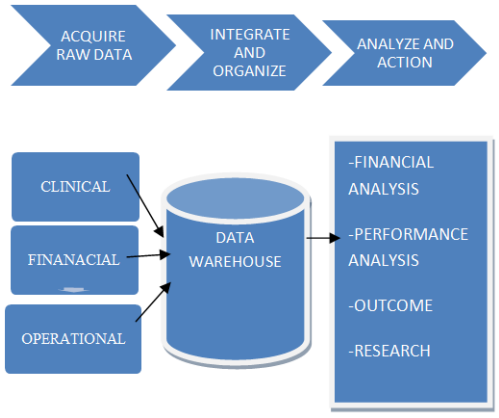

Figure 1. Database Forensic Model

Data in the digital forensic investigation is viewed in four steps [10]:

- *In the form of tree, data to be analyzed is viewed. The* critical data structure can be the root from which all other data can be reached.
- Metadata can be used to locate all data objects [4]. Metadata refers to data about the data that is stored within a source of digital evidence [11]. There are many levels at which metadata can be defined. They are system metadata, file system metadata, application metadata, document metadata, email metadata, business metadata, geographical metadata etc. A particular type of metadata provides information at certain context. The information enables in easy handling and management of the data contained.
- Database stores information regarding each data object. In-memory databases are used by some tools, and others use external SQL databases.
- Carving is used by some tools to locate data objects that cannot be reached from the root. Deleted data or partial file data is useful in investigation which gave rise to the new field of data carving. Carving is the process of identifying the file types using a string of bytes, called magic numbers, from an image and matching with a database of known magic numbers to recover deleted or partially deleted files[11].

Forensic analysis can be done of table storage, transaction logs and indexes [5].

In B+-trees indexes, disk representation reveals expired data and information about history of operation. Today's database systems are more prone to modifications due to the huge amount of data. Engines like InnoDB for MySQL stores manipulations statement in log files. The traditional database forensic approach becomes ineffective if someone removes the tracks of manipulations from log files. B+-Trees are used to handle large amounts of data. B+-Trees can detect manipulations of data by a malicious administrator. A database is organized to support updates, retrieval, and management the data. Databases must support operations, such as retrieval, deletion and insertion of data. Databases are usually large and cannot be maintained entirely in memory. B+-trees are used to index the data and to provide fast access. We will deal with forensic aspects of B+-Trees.

Thus, we tried to relate trees and database forensics. Some work has been done in database forensic area using B+-trees data structure.

II. Related Work

Database Forensic is an important area which must need special research attention. A good amount of work is not done due to the complexity of databases. From forensic point of view databases are inherently multidimensional. The paper "InnoDB Database Forensics" [6] shows that tables were built in the .frm files format in MySQL. The main purpose was to determine and detect database inconsistencies. The various researchers have worked in the area of database forensic and the summary of their work is described in the paper "Database Security Threats and Challenges in Database Forensic" [2]. The research paper on "Analysis of B-tree data structure and its usage in computer forensics" [9] summarizes that B-tree is a fast indexing method in which indexes are organized into a set of nodes, where each node contains indexed data. Nodes are multilevel. B-tree is used in databases and file systems. Numbers of disc accesses are reduce by using B-trees. B-tree is reorganized after deletion of files or directories and entries of the deleted value is overwritten. It appears to be deleted but actually is overwritten. The main disadvantage described in this paper is that B-trees that there seems to be a temporary intermediate state where the entry is present in tree but is rendered invalid. The work on "Using the Structure of B+-Trees for Enhancing Logging Mechanisms of Databases" [7] describes the various methods to prevent unauthorized access and modifications in database management systems. But still modifications can be made in the database. By using signature of a B+-Tree, the logging mechanism in databases is supported. By defining the signature of a B+-Tree, the actual structure of a B+-Tree is stored separately from the data. Using transaction logs, changes in the data can be retrieved. Thus, on each transaction, the structure of tree is compared with the tree signature. This mechanism is useful for forensic analysis applied to the underlying B+-Tree-structure of an index. This mechanism can be very useful in the case of defining a Forensic-Aware database. We can easily reconstruct the old versions of the tree. The limitation of this approach is that it is expensive. When each changes in the underlying structure is logged, only than transaction is assured. This will result in some additional space needed, as well as an additional logging operation after each transaction. The paper "Trees cannot lie: using data structures for forensics purposes"[8], put forward the possibilities of using B+-trees data structure in database forensic. Although database forensic lie in its generalization which should be applicable for all types of operations, B+-tree does not give much information about this is the disadvantage described in the paper.

III. Database Forensic and B+-Trees

A B^+-tree comes under the family of multi-way search trees. These trees were first proposed by Bayer and McCreight in 1972. It has replaced almost all large file access methods other than hashing. These multi-way balanced search trees are now the standard file organization for applications requiring insertion, deletion, and key range searches. They have the following advantages:

- *B*-trees are always height balanced, with all leaf nodes at the same level
- Update and search operations affect only a few disk pages, so performance is good.
- B-trees keep related records on the same disk page, which takes advantage of locality of reference.
- B-trees guarantee that every node in the tree will be full at least to a certain minimum percentage. This improves

space efficiency while reducing the typical number of disk fetches necessary during a search or update operation over many thousands of records.

A B+-tree is a generalization of a binary search tree (BST) in some aspects. The nodes of a B+-tree point to many children nodes. It minimizes disk accesses whenever we are trying to locate records. A B+-tree of order *m* is a tree where each internal node contains up to *m* branches i.e. children nodes and thus store up to *m*-search key values in a BST, only one key value is needed. At the leaf nodes, the B+-tree stores records and pointers to actual records and they all are a found at the same level in the tree. Therefore the tree is always height balanced.

The time to retrieve from external memory is thousands of times greater than high-speed memory. To minimize the disk access is the main goal.

Fast access to data records is very important in database. In today's computer systems, I/O operations are carried out very slowly. We will focus on decreasing the amount of time required for input output operations. In modern database storage engines, use of index allows fast lookup of records with a small amount of I/O operations. In InnoDB which is a storage engine, the index is builds up a B+-Trees.

The index generates a highly structured tree. B+-trees have performance advantages by reducing expensive I/O operations as compared to B-trees. As I/O operations are reduced, the time required for detecting anomalies in the database is less.

A. B^+-tree operations

To understand the B^+-tree operations more clearly, assume that there is a table whose primary is a single attribute and that it has a B^+-tree index organized on the PK attribute of the table.

1) Searching

To retrieve records, queries are written with conditions that describe the values that the desired records are to have. The most basic search on a table to retrieve a single record given its PK value K.

Search in a B^+-tree is an alternating two-step process, beginning with the root node of the B^+-tree. Say that the search is for the record with key value K -- there can only be one record because we assume that the index is built on the PK attribute of the table.

- Perform a binary search on the search key values in the current node -- recall that the search key values in a node are sorted and that the search starts with the root of the tree. We want to find the key Ki such that $Ki \leq K < Ki+1$.
- If the current node is an internal node, follow the proper branch associated with the key Ki by loading the disk page corresponding to the node and repeat the search process at that node.
- If the current node is a leaf, then*:*
    - If $K=K_i$, then the record exists in the table and we can return the record associated with K_i
    - Otherwise, K is not found among the search key values at the leaf, we report that there is no record in the table with the value K.

2) Inserting into a B^+-tree

Insertion in a B^+-tree is similar to inserting into other search trees, a new record is always inserted at one of the leaf nodes. The complexity added is that insertion could overflow a leaf node that is already full. When such overflow situations occur a brand new leaf node is added to the B^+-tree at the same level as the other leaf nodes. The steps to insert into a B^+-tree are:

- *Follow the* path that is traversed as if a Search is being performed on the key of the new record to be inserted.
- The leaf page L that is reached is the node where the new record is to be indexed.
- If L is not full then an index entry is created that includes the search key value of the new row and a reference to where new row is in the data file. We are done; this is the easy case!
- If L is full, then a new leaf node Lnew is introduced to the B+-tree as a right sibling of L. The keys in L along with the an index entry for the new record are distributed evenly among L and Lnew. Lnew is inserted in the linked list of leaf nodes just to the right of L. We must now link Lnew to the tree and since Lnew is to be a sibling of L, it will then be pointed to by the parent of L. The smallest key value of Lnew is copied and inserted into the parent of L -- which will also be the parent of Lnew. This entire step is known as commonly referred to as a split of a leaf node.
    - If the parent P of L is full, then it is split in turn. However, this split of an internal node is a bit different. The search key values of P and the new inserted key must still be distributed evenly among P and the new page introduced as a sibling of P. In this split, however, the middle key is moved to the node above -- note, that unlike splitting a leaf node where the middle key is copied and inserted into the parent, when you split an internal node the middle key is removed from the node being split and inserted into the parent node. This splitting of nodes may continue upwards on the tree.
    - When a key is added to a full root, then the root splits into two and the middle key is promoted to become the new root. This is the only way for a B+-tree to increase in height -- when split cascades the entire height of the tree from the leaf to the root.

3) Deletion

ISSN: 2321-8169
1713 – 1717

Deletion from a B^+-tree again needs to be sure to maintain the property that all nodes must be at least half full. The complexity added is that deletion could underflow a leaf node that has only the minimum number of entries allowed. When such underflow situations take place, adjacent sibling nodes are examined; if one of them has more than the minimum entries required then, some of its entries are taken from it to prevent a node from "under-flowing". Otherwise, if both adjacent sibling nodes are also at their minimum, then two of these nodes are merged into a single node. The steps to delete from a B^+-tree are:

- Perform the search process on the key of the record to be deleted. This search will end at a leaf L.
- If the leaf L contains more than the minimum number of elements (more than m/2 - 1), then the index entry for the record to be removed can be safely deleted from the leaf with no further action.
- If the leaf contains the minimum number of entries, then the deleted entry is replaced with another entry that can take its place while maintaining the correct order. To find such entries, *we inspect the two sibling leaf nodes* L_{left} *and* L_{right} *adjacent to L -- at most one of these may not exist.*
    - If one of these leaf nodes has more than the minimum number of entries, then enough records are transferred from this sibling so that both nodes have the same number of records. This is a heuristic and is done to delay a future underflow as long as possible; otherwise, only one entry need be transferred. The placeholder key value of the parent node may need to be revised.
    - If both L_{left} and L_{right} have only the minimum number of entries, then L gives its records to one of its siblings and it is removed from the tree. The new leaf will contain no more than the maximum number of entries allowed. This merge process combines two subtrees of the parent, so the separating entry at the parent needs to be removed -- this may in turn cause the parent node to underflow; such an underflow is handled the same way that an underflow of a leaf node.
    - If the last two children of the root merge together into one node, then this merged node becomes the new root and the tree loses a level.

IV. Proposed Work

Figure 2 shows the system architecture of the tool in which we are using B+-trees to detect anomalies in the database. By using this tool we can detect modification made in the database [3]. In our approach we will design a tool that first identify and collect the databases i.e. text files, binary logs and log files. Parsers are used to read huge files. Then metadata is formed by collecting raw data. To build inference rules for making decisions to extract useful information from metadata, data mining techniques are used.

Then by using the theorem: let B be a B+-Tree with n > m elements which are added in ascending order. Then it holds true that the partition of the leafs of B has the following structure [8]:

$n = \sum_{j=1}^{p} b_j$, *with* b_j= *m/2+1,* $\forall j \neq p$ *and* $b_p \geq m/2$.

According to the theorem above, B+-tree structure for the database is created. When we start inserting elements in B+-tree, we enter element into an empty root, on inserting the next element we have to split the root and generate a new one with two leafs. The new element is added to the rightmost leaf. So the rightmost leaf contains m/2+1 element and leftmost contains m/2 elements. Fill rate of the tree structure is stored.

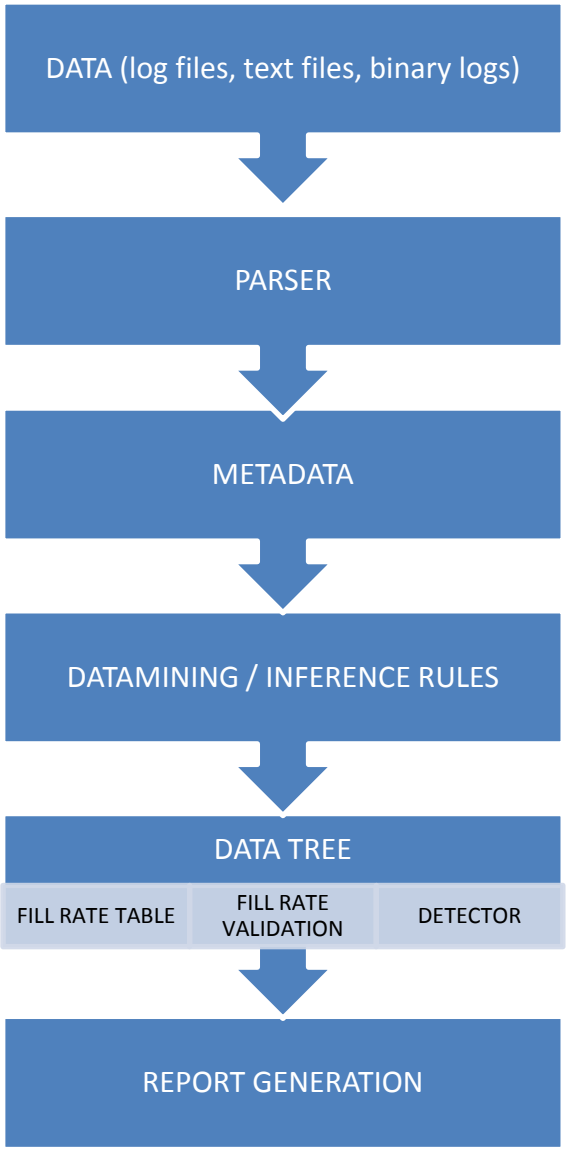

Figure 2 System Architecture of a tool which uses B+-trees to detect anomalies in the database.

If a malicious administrator inserts a forged record, it would then reside in the leaf that contains m/2+1 element. The fill rate of leaf nodes in this tree is analyzed, and it will be found that the record is located in a node that has a too high

fill rate for strictly sorted inserts (i.e. >m/2+1 elements). Thus modifications can be detected. Similarly, deletion of the record can also be detected by checking fill rates if the number of elements on the leaf node appears below m/2. Report is generated of the anomalies detected which can be useful in database forensics.

CONCLUSION

Database Forensic is a very new field which needs more attention. The organizational data is very crucial. Even minor modification in the database can result in great damage to the organization. There are some tools available which are used in organizations to provide security to the database from any kind of tampering. Effective implementations of search trees are necessary for any databases. B+-Trees can be used effectively in database forensic. We have chosen B+-Tree as it is one of the most widely used data structure and is a fast indexing data retrieval technique. In our approach we will use B+ -Tree to give strong forensic evidence for many cases of retroactive manipulation in tables, thus providing the investigator with the new tool. We will focus on increasing the efficiency of the tool by reducing expensive I/O operations.

REFERENCES

[1] B. Ooi and K. Tan, "B-trees: bearing fruits of all kinds," in Proceedings of the 13th Australasian database conference-Volume 5. Australian Computer Society, Inc., 2002, pp. 13–20. 285

[2] Harmeet Kaur Khanuja and Dr. D. S. Adane (2011), "Database Security Threats and challenges in Database Forensic: A survey", Proceedings of 2011 International Conference on Advancements in Information Technology (AIT 2011), available at http://www.ipcsit.com/vol20/33-ICAIT2011- A4072.pdf

[3] Harmeet Kaur Khanuja and Dr. D. S. Adane (2011), "A FRAMEWORK FOR DATABASE FORENSIC ANALYSIS", An International Journal (CSEIJ), Vol.2, No.3, June 2012

[4] Martin S. Olivier. (2009, March), "On metadata context in Database Forensics, Digital Investigation", Elsevier, www.sciencedirect.com, Volume 5, Issues 3-4, Pages 115-123.

[5] Patrick Stahlberg, Gerome Miklau, and Brian Neil Levine, "Threats to privacy in the forensic analysis of database systems", SIGMOD'07, June 11–14, 2007, Beijing, China.

[6] Peter Frühwirt, Markus Huber, Martin Mulazzani, Edgar R. Weippl, "InnoDB Database Forensics", Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference, April 2010.

[7] Peter Kieseberg Sebastian Schrittwieser Lorcan Morgan Martin Mulazzani Markus Huber Edgar Weippl, "Using the Structure of B+-Trees for Enhancing Logging Mechanisms of Databases", SBA Research Vienna, Austria, 2011.

[8] Peter Kieseberg, Sebastian Schrittwieser, Martin Mulazzani, Markus Huber and Edgar Weippl, "Trees Cannot Lie: Using Data Structures for Forensics Purposes", European Intelligence and Security Informatics Conference, 2011

[9] P. Koruga and M. Baˇca, "Analysis of B-tree data structure and its usage in computer forensics," in Central European Conference on Information and Intelligent Systems, 2010.

[10] Simson L. Garfinkel, "Digital forensics research: The next 10 years", Digital Forensic Research Workshop, Elsevier Ltd, 2010.

[11] Sriram Raghavan,"Digital forensic research: current state of art", CSIT (March 2013) 1(1):91–114.

[12] http://en.wikipedia.org/wiki/Digital_forensics.